

Code. Runs in Visual Basic 6, an abandoned programming language. Executable on request.

```
-----

'Calculate the maximum size of the effect of a zero-weighted constraint
'This uses the method of running many random trials.

Option Explicit

'Module-level variables.

    Dim mNumberOfTrials
    Dim mLowestDifference As Single      'I.e., difference between the weight of PreferMajority and PreferMinority
    Dim mHighestDifference As Single     'The program explores a range of differences.
    Dim mIncrement As Single            'The fine-grainedness of the search.

Private Sub Command1_Click()

    'Top routine, runs when user clicks on a button on a little interface, not seen here.

    'Reuse the Go button as an Exit button.
        Static ButtonStatus As Boolean
        If ButtonStatus Then
            Close
            End
        Else
            Let ButtonStatus = True
        End If

    'Call the various routines.
        Call SetParameters
        Call OpenOutputFile
        Call RunTrialsOuter

    'Announce completion.
        Let Command1.Caption = "Done! Click again to exit."
        DoEvents

End Sub
```

```
Sub SetParameters()
```

```
    'Whatever is not commented out, runs.
```

```
    'Let mNumberOfTrials = 10000000  
    Let mNumberOfTrials = 100000
```

```
    'Very coarse version:  
        'Let mLowestDifference = -12  
        'Let mHighestDifference = 6  
        'Let mIncrement = 0.25
```

```
    'Coarse version:  
        'Let mLowestDifference = -2  
        'Let mHighestDifference = 1  
        'Let mIncrement = 0.1
```

```
    'Fine version:  
        'Let mLowestDifference = -0.7  
        'Let mHighestDifference = -0.5  
        'Let mIncrement = 0.005
```

```
    'Yet another version:  
        Let mLowestDifference = -3  
        Let mHighestDifference = 3  
        Let mIncrement = 0.005
```

```
End Sub
```

```
Function OpenOutputFile()
```

```
    If chkEnforceClipping.Value = vbChecked Then  
        Open App.Path + "/ResultsComputedByTheProgram_Clipping.txt" For Output As #2  
    Else  
        Open App.Path + "/ResultsComputedByTheProgram_NoClipping.txt" For Output As #2  
    End If
```

```
    'Give the basic parameters up front.
```

```
        Print #2, "Weight difference ranged from " + Trim(Str(mLowestDifference)) + " to " +  
Trim(Str(mHighestDifference)) + " with increment " + Trim(Str(mIncrement)) + "."  
        Print #2, "There were " + Trim(Str(mNumberOfTrials)) + " trials for each weight difference."
```

```

    If chkEnforceClipping.Value = vbTrue Then
        Print #2, "Clipping was enforced."
    Else
        Print #2, "Clipping was not enforced."
    End If
    Print #2, "There were " + Trim(Str(mNumberOfTrials)) + " trials for each weight difference."

'Column labels
    Print #2,
    Print #2, "MajHelper viols";
    Print #2, Chr(9);
    Print #2, "Wght. diff.";
    Print #2, Chr(9);
    Print #2, "P-unhelped";
    Print #2, Chr(9);
    Print #2, "P-helped";
    Print #2, Chr(9);
    Print #2, "Diff";
    Print #2,

End Function

Sub RunTrialsOuter()

    'We try various values of MajorityHelperViolations. For our paper, end of section 3.1, this should be 1. For
    section 3.2, "largest possible effect", we need a large value.

    Dim MajorityHelperViolations As Long

    'Whatever is not commented out. You can run more than one if you like.
    Let MajorityHelperViolations = 1
    'Let MajorityHelperViolations = 1000
    'Let MajorityHelperViolations = 100000
    'Let MajorityHelperViolations = 10000000

    Call RunTrials(MajorityHelperViolations)

End Sub

```

```
Sub RunTrials(MajorityHelperViolations As Long)
```

```
    Dim CurrentDifference As Single      'Difference in weights of MajorityHelper and MinorityHelper
    Dim UnhelpedMajorityWins As Long    'Counter for Input1; not "helped" by MajorityHelper.
    Dim HelpedMajorityWins As Long      'Counter for Input2; "helped" by MajorityHelper.
    Dim TrialIndex As Single             'Counter for the number of trials in the random sample
```

```
    'We loop through values of CurrentDifference, which is the weight difference between MajorityHelper and
    MinorityHelper.
```

```
    '    This loop is defined by mLowestDifference, mHighestDifference, and mIncrement.
```

```
    'Outer loop -- loop through differences, starting with lowest value.
```

```
        Let CurrentDifference = mLowestDifference
```

```
        Do
```

```
            'Report progress
```

```
                Let Command1.Caption = "MH viols: " + Trim(Str(MajorityHelperViolations)) + "  lowest: " +
Trim(Str(mLowestDifference)) + "  highest: " + Trim(Str(mHighestDifference)) + "  now: " +
Trim(Str(Round(CurrentDifference, 3)))
```

```
                DoEvents
```

```
            'Initialize the counts of wins
```

```
                Let UnhelpedMajorityWins = 0
```

```
                Let HelpedMajorityWins = 0
```

```
            'Compute unhelped and majority wins, with the requested number of trials with these weights
```

```
                For TrialIndex = 1 To mNumberOfTrials
```

```
                    'Call the function TrialOutcome() to see who wins on this trial. "True" means MajorityHelper is
violated.
```

```
                        '    Increment the counts with what you find.
```

```
                            Let UnhelpedMajorityWins = UnhelpedMajorityWins + TrialOutcome(CurrentDifference,
MajorityHelperViolations, False)
```

```
                            Let HelpedMajorityWins = HelpedMajorityWins + TrialOutcome(CurrentDifference,
MajorityHelperViolations, True)
```

```
                Next TrialIndex
```

```
            'Print the result
```

```
                Print #2, Trim(Str(MajorityHelperViolations));
```

```
                Print #2, Chr(9);
```

```
                Print #2, Round(CurrentDifference, 3);
```

```
                'Percentage of victory for the majority candidate is found by dividing its win count by the number of
trials.
```

```
                    Print #2, Chr(9);
```

```
                    Print #2, Round(UnhelpedMajorityWins / mNumberOfTrials, 4);
```

```
                'Ditto when it is helped by MajorityHelper.
```

```

        Print #2, Chr(9);
        Print #2, Round(HelpedMajorityWins / mNumberOfTrials, 4);
        'Also compute the difference -- the effect, in this case, of Majority Helper.
        Print #2, Chr(9);
        Print #2, Round((HelpedMajorityWins - UnhelpedMajorityWins) / mNumberOfTrials, 4);
        'End of line
        Print #2,
        'Augment the weight difference and exit if you've surpassed the upper limit.
        Let CurrentDifference = CurrentDifference + mIncrement
        'Differences not exact so give .001 leeway.
        If CurrentDifference > mHighestDifference + 0.001 Then Exit Do
Loop

    'Print a blank line to help with plotting
    Print #2,

```

End Sub

Function TrialOutcome(CurrentDifference As Single, MajorityHelperViolations As Long, Helped As Boolean) As Long

```

    'Compute the winning candidate on a single random trial, depending on weight difference between Prefer Majority and
    '    and Prefer Minority, and on whether Majority Helper is violated.

    Dim DiagnosticGaussian      'See ReadMe.pdf of the Supplemental Materials for discussion of diagnostic
Gaussians.
    Dim MyGaussian              'A random sample from the Gaussian distribution, mean zero and standard deviation 1.

    'The mean of the diagnostic Gaussian is set at the mean of the weights of the constraints favoring one
candidate, minus the means of the
    '    weights favoring the other.
    Let DiagnosticGaussian = CurrentDifference
    'The Gaussian of PreferMajority
    Let DiagnosticGaussian = DiagnosticGaussian + Gaussian
    'The Gaussian of PreferMinority. Subtraction is symbolic only, since the Gaussian distribution is symmetrical
about zero.
    Let DiagnosticGaussian = DiagnosticGaussian - Gaussian
    'The truncated Gaussian of Majority Helper, if appropriate.
    If Helped Then
        Let MyGaussian = Gaussian
        If chkEnforceClipping.Value = vbChecked Then

```

is positive. 'Enforce Clipping. Since the weight of Majority Helper is zero, we simply check if the perturbation

majority candidate win. If MyGaussian > 0 Then
 Let DiagnosticGaussian = DiagnosticGaussian + (MajorityHelperViolations * MyGaussian)
 'Note that where MajorityHelperViolations is huge, this value will suffice to make the

 Else
 'do nothing: because of truncation, there is nothing to add to the DiagnosticGaussian.
 End If

 Else
 'Assume no clipping.
 Let DiagnosticGaussian = DiagnosticGaussian + (MajorityHelperViolations * MyGaussian)
 End If
End If

'Return 1 (Majority candidate wins) if the diagnostic Gaussian is greater than zero, else zero (Minority candidate wins).

 If DiagnosticGaussian > 0 Then
 Let TrialOutcome = 1
 Else
 Let TrialOutcome = 0
 End If

End Function

Function Gaussian() As Single

 'This algorithm for producing random values from the Gaussian
 ' distribution gives you two values at once. The static variables
 ' below remember the second value, and that you have one available.

 'Thanks to Paul Boersma, long ago, for providing me with this algorithm.

 Dim fac As Single, r As Single, v1 As Single, v2 As Single
 Static blnValuedAlreadyStored As Boolean
 Static StoredValue As Single

 If blnValuedAlreadyStored = False Then
 'Basic calculations:
 Do

```

        'Call random numbers and convert to a -1 to 1 range.
        Let v1 = 2 * Rnd - 1
        Let v2 = 2 * Rnd - 1
        Let r = v1 * v1 + v2 * v2
        'The following condition guarantees that the output will have
        ' summed squares less than one. This happens about 70% of the
        ' time, so you're pretty much guaranteed to get out fairly soon.
        If r < 1 And r > 0 Then Exit Do
    Loop
    'The following yields one normal deviate. Save it for next time.
    Let fac = Sqr(-2 * Log(r) / r)
    'Let StoredValue = 2 * v1 * fac          'Boersma likes a s.d. of two.
    Let StoredValue = v1 * fac
    'This computes the other normal deviate, which can be used fresh.
    'Let Gaussian = 2 * v2 * fac
    Let Gaussian = v2 * fac
    'Use the flag to note that you don't have to compute a new one next time.
    Let blnValuedAlreadyStored = True
Else
    'Return the thriftily stored value.
    Let Gaussian = StoredValue
    'Indicate with the flag that you've used it up and must compute anew next time.
    Let blnValuedAlreadyStored = False
End If      'Should I compute a new value?

End Function

```